TITLE OF THE INVENTION

A METHOD FOR EFFICIENTLY COMPUTING A FAST FOURIER TRANSFORM

CROSS REFERENCE TO RELATED APPLICATIONS

N/A

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

N/A

BACKGROUND OF THE INVENTION

This invention relates generally to a technique for computing a Fast Fourier Transform (FFT) and more particularly to methods and apparatus for computing an FFT in which the number of loop operations are reduced and the resultant output data values from each stage data are stored in a memory with a unity stride.

The fast Fourier transform (FFT) is the generic name for a class of computationally efficient algorithms that implement the discrete Fourier transforms (DFT), and are widely used in the field of digital signal processing.

A band-limited time-varying analog signal can be converted into a series of discrete digital signals by sampling the analog signal at or above the Nyquist frequency, to avoid aliasing, and digitizing the sampled analog signals. A DFT algorithm may be applied to these digitized samples to calculate the discrete frequency components contained within the analog signal. The DFT algorithm provides, as output data values, the magnitude and phase of the discrete frequency components of the analog signal. These discrete frequency components are evenly spaced between 0 and ½ the sampling frequency, which is

Express Mail Number
EL418425050US

typically the Nyquist sampling frequency. The number of discrete frequency components is equal to the number of the digitized samples that are used as input data. For example, a DFT having 8 input samples, will have 8 evenly spaced frequency components as output.

The DFT is given by:

$$X(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)e^{j\frac{2\pi nk}{N}}$$

where:

N is the number of input samples;

n is the particular index in the time domain sample from n=0 to n=N-1;

x(n) is the magnitude of the time domain analog signal at the time sample point corresponding to n;

k is the particular frequency domain component from k=0 to k=N-1; and

X(k) is the magnitude of the frequency component corresponding to the frequency index k.

The DFT involves a large number of calculations and memory operations and, as such, is not computationally efficient. The FFT algorithm reduces the computational load of calculating the discrete frequency components in a time domain signal from approximately $6(N^2)$ to approximately $N\log_2 N$. As will be discussed in detail below, this reduction in the number of calculations is achieved by decomposing the standard DFT algorithm into a series of smaller and smaller DFTs. For example, an 8 point DFT can be decomposed into an FFT involving 3 stages of calculations. In this manner the 8 point FFT is decomposed into one 8 point FFT that can be decomposed into two 4 point DFTs that are decomposed into four 2 point DFTs.

At each stage of the FFT algorithm the canonical mathematical operations performed on each pair of input data is known as the FFT butterfly operation. Figure 4 illustrates the canonical FFT butterfly operations which are

$$X(m+1) = X(m) + W(n,k)Y(m)$$
$$Y(m+1) = X(m) - W(n,k)Y(m)$$

where X and Y are input signals and are discussed in more detail below. W(n,k) (the "twiddle factor") is a complex value and is given by the formula:

$$W(n,k) = e^{j\frac{2\pi nk}{N}}.$$

This complex function is periodic and for an FFT of a given size N, provides N/2 constant values. As discussed in more detail below, these values may be pre-calculated and stored in a memory.

Figure 1 illustrates a traditional decimation in time FFT signal flow graph for an 8 input (8 point) FFT. An FFT algorithm will include $\log_2 N$ stages of calculations. Thus, the 8 point FFT signal flow graph 100 is divided into $\log_2 8$, or three, stages: the first stage 102, second stage 104 and the third stage 106, where each stage performs N/2 butterfly calculations. Thus, in Fig. 1, every stage of the signal flow graph 100 will calculate 8/2, or 4 butterfly calculations per stage. An examination of Fig.1 also shows that the first stage provides four 2 point FFTs, the second stage provides two 4-point FFT's and the final stage provides one 8-point FFT. Thus, each stage will have a number of groups in which the FFTs are calculated. The number of groups per stage is given by:

$$groups = 2^{Log_2(N)-m}$$

where N is the number of input data points, and m is the number of the stage and is from m=1 to $m=\log_2 N$. Thus in

Fig. 1, the first stage has 2^{3-1} or 4 groups of FFTs, 108, 110, 112, and 114. The second stage has 2^{3-2} or 2 groups of FFTs, 116, and 118. The final stage has 2^{3-3} or 1 group of an FFT 120.

To compute an FFT on a computer, the signal flow graph 100 must be translated into a software program. A software program based on the traditional FFT signal flow graph will first typically re-order the data into a bit-reversed order as shown by the input data 122. Next, three loops that calculate FFT data are executed. The outermost loop, known as the stage loop, will be executed only for each stage. Therefore, for an N point FFT, there will be $\text{Log}_2 N$ outer loops that must be executed. The middle loop, known as the group loop, will be executed a different number of times for each stage. As discussed above, the number of groups per stage will vary from $2^{\log_2 (N)-m}$ to 1 depending on the position of the stage in the algorithm. Thus for the early stages of the FFT the group loop will be entered into and out of many times in each stage. The inner most loop, known as the butterfly loop, will be executed N/2 times for each stage.

The FFT signal flow graph 100 also illustrates another aspect of the traditional FFT technique. The data that is provided by each butterfly calculator is stored in a different sequence in each stage of the FFT. For example, in the first stage 102 the input data is stored in a bit-reversed order. Thus, each butterfly calculator receives input data values that are stored in adjacent memory locations. In addition, each butterfly calculator provides output data values that are stored in adjacent memory locations in the sequence in which they are calculated. In the second stage 104 each butterfly calculation receives input data that is separated by 2 storage locations, and the output data values are stored in memory locations that are also 2 storage locations apart. In the third stage 106, each butterfly calculation receives data that is 4 storage locations apart and provides output data values that are also stored 4 storage locations apart. Thus, the distance between the storage locations where the output data values are stored (the stride) varies as a power of two from 2^0 to $2^{N/2}$. Thus, in the illustrative embodiment the stride varies between 1 and 4 as discussed above.

In a typical computing system, the most time consuming operations are the reading and writing of data to and from memory respectively. Since the FFT is a very data intensive algorithm, many schemes have been developed to optimize the memory-addressing problem. Typically memory systems have been designed to increase the performance of the FFT by changing the pattern of how the memory is stored, by using smaller faster memories for the data, or by dedicating specific hardware to calculate the desired memory locations. However, the very nature of the traditional FFT as shown in Fig. 1 illustrates the limitations of these approaches. For each stage, a new stride will have to be computed and, for each stage, there are only so many ways to change the pattern of the memory storage. Modern computer languages also allow the accessing of memory locations directly using "pointers". Pointer arithmetic can be time consuming as well and the need to recalculate the pointer arithmetic for each stage is inefficient.

In addition to the data storage problem, traditionally, the control and overhead processing for a computer program takes up the bulk of the program memory, but only a small fraction of the actual processing time. Therefore, minimizing the control and overhead portions of a computer program is one method to further optimize the memory usage of the program. As discussed above, in the signal flow graph 100 the number of the stage loops and the butterfly loops to be executed are set by the system parameters, in particular the number of input data points used. The number of group loops to be executed however changes with each stage. In particular, in the early stages of the algorithm, the overhead and control software will be executing a large number of group loops each having a small number of butterfly loops for each stage. This entering and exiting of the group loops will result in a complex iteration space in which a large number of overhead and control instructions need to be executed, resulting in an inefficient program execution.

It would therefore be desirable to be able to compute an FFT in a manner that reduces the number of required iterations and simplifies the calculation of the storage locations of the output data values from each stage in memory.

BRIEF SUMMARY OF THE INVENTION

The present invention provides a method and apparatus for computing a FFT in which a unity stride is used to store the ouput data from each stage, and each stage of the FFT does not include a group loop calculation stage.

A method for computing an FFT is disclosed in which a sequence of first data points is received and stored in a first memory area. An FFT butterfly calculator selects R input data from the sequence of first data points where the input data are separated by N/R data points. The FFT butterfly calculator also receives the appropriate twiddle factors that are stored in sequential locations in a bit reversed order in a second memory area. The FFT butterfly calculator calculates a radix R butterfly calculation and stores the output data values in a third memory in the sequence in which they are calculated.

BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWING

The invention will be more fully understood from the following detailed description taken in conjunction with the accompanying drawings in which:

Fig. 1 is a schematic illustration of a signal flow diagram of a traditional FFT;

Fig. 2 is a block diagram illustrating an illustrative embodiment of the present invention;

Fig. 3 is a signal flow diagram illustrating the signal flow diagram of an exemplary FFT calculation stage of the present invention;

Fig. 4a is a signal flow diagram illustrating an FFT butterfly calculator;

Fig. 5 is a graphical illustration of the twiddle factors for an 8 point FFT;

Fig. 6 is a signal flow diagram for an 8 point FFT consistent with the present invention; and

Fig. 7 is a table illustrating the storing of twiddle factors in bit reversed order.

DETAILED DESCRIPTION OF THE DRAWINGS

A method consistent with the present invention for calculating a fast Fourier transform (FFT) more efficiently than in traditional methods is disclosed. Fig. 2 depicts a block diagram of a system operative in a manner consistent with the present invention. Input data values 202 are received and written into a first memory area 204. As will be explained in detail below, unlike traditional FFT methods, no re-ordering of the input data values 202 occurs. The FFT calculator stage 208 retrieves the input data values 202 needed for the calculation and also retrieves the twiddle factor values stored in bit-reverse order in a second memory area 210. FFT calculation stage 208 calculates output data values 206. These output data values are stored sequentially in a third memory area 212, in the order in which they are calculated. As will be explained in detail below, this ordering and storing of the output data values in the order of calculation provides for unity stride memory operations for the output data values.

A loop controller 214 provides the overhead and process control functions for the algorithm. The loop controller 214 monitors which stage is currently executing, and determines which data are required. If more stages still need to be executed, the output data values 210 stored in the third memory area 212 is used as the input data values 202 for the next stage. If are no more stages to be executed, then the output data values 212 represents the discrete frequency components of the original band-limited analog data.

Fig. 3 illustrates one embodiment of a signal flow diagram of an FFT calculator stage 300 consistent with the presently disclosed FFT method. The signal flow diagram illustrating the FFT calculator stage 300 will be identical for any stage of the presently disclosed FFT method. Thus, every stage of the FFT method consistent with the presently disclosed FFT method will have a signal flow diagram geometry that is the same as every other stage.

In the signal flow graph, input data values 302 are provided to the FFT calculator stage 300. The input data 302 is stored sequentially in the time-order in which each sample is taken. Thus, the input data values 302 represent the sampled and digitized band-limited analog signal in the sequential order in which they were sampled. This is in contrast to the bit-reversed storage required for the traditional FFT algorithm illustrated in Fig. 1. Storing the data sequentially results in a more efficient FFT algorithm because simpler calculations for the memory operations are required for each stage. As discussed above, the fewer memory operations a computer program executes, the more efficient it will be.

The number of input and output data values the FFT butterfly calculator has determines the radix of an FFT. Therefore for the embodiment illustrated in Fig. 3, each FFT butterfly calculator in the FFT calculator stage 300 has two inputs and is a radix 2 FFT. An FFT method that has 4 inputs for each FFT butterfly calculator is a radix 4 FFT. Although the illustrative embodiment described herein is for a radix 2 FFT, it would obvious to modify the presently disclosed FFT method for an FFT having radixes other than 2 and for a FFT having a mixed-radix as well.

Each of the FFT butterfly calculators in the FFT calculator stage 300 shown in Fig 3 requires, as input, two data input values and one twiddle factor value and provides, as output, two output data values. In each stage of the presently disclosed FFT method the input data values for any FFT butterfly calculator are N/R data points apart. Where N is the number of input data points and R is the radix of the FFT. In the embodiment illustrated in Fig. 3, which is an 8-point radix-2 FFT, the two input data values will be N/2, or 4, data points apart.

Output stage 308 illustrates the storage of the output data values in the order in which it is calculated. The first butterfly calculator uses the input data x(0) and x(4). The output data values from this FFT butterfly are stored in the first and second memory locations 310, 312. The second butterfly calculator uses the input data x(1) and x(5). The output data values from this FFT butterfly are stored in the third and fourth memory locations 314, 316. The third butterfly calculator uses the input data x(2) and x(6). The output data values from this FFT butterfly are stored in the fifth and sixth memory locations 318, 320. The fourth butterfly calculator uses the input data x(3) and x(7). The output data values from this FFT butterfly are stored in the seventh and eighth memory locations 322, 324. Thus, the output data values are "re-ordered" according to the calculation order, unlike the traditional FFT algorithm illustrated in Fig.1.

The re-ordering of the output data values, such that they are written into memory with a unity stride, helps to increase the efficiency of the presently disclosed FFT method. The unity stride reduces the number of calculations needed for the memory operations and, in addition, simplifies the arithmetic when using pointers or other memory accessing functions.

Fig. 4 illustrates a suitable radix 2 FFT butterfly calculator 400 consistent with the presently disclosed FFT method. Input data Y(m) 402 and X(m) 404 are mathematically manipulated and combined to provide output data values 406 and 408. Input data Y(k) 404 is multiplied, by multiplier 409, with W_N^{nk} 410. W_N^{nk} is one of a predetermined number of "twiddle factor" constants that are used throughout the FFT method. Thus, for each stage in an FFT, there will be N/2 twiddle factors required. However, as will be explained below, there will not in general be N unique twiddle factors per stage.

In general, the twiddle factors are complex numbers having real and imaginary parts. The resultant product, which in general is also complex, is added, by adder 412, to input data X(m) 402 to form the output data value 406, and subtracted, by adder 414, from X(m) to form the output data value 408. Thus, in general, the arithmetic of the butterfly calculator is complex and requires complex additions and multiplications.

The twiddle factor data is required for every stage of the FFT method. As shown in figure 5, the twiddle factor values can be illustrated as points that are equidistantly and symmetrically spaced apart on the circumference of the unit circle in the imaginary plane. Thus, each twiddle factor represents a magnitude and a phase since it has both real and imaginary parts. In the presently disclosed FFT method, the number of twiddle factor values is equal to one-half the number of input data values for each FFT butterfly calculator. Thus, Fig. 5 represents the twiddle factors for an 8 point FFT. However, as is illustrated in Fig. 5, the twiddle factors are symmetric with respect to the origin. Thus, each twiddle factor will be the negative of the real and imaginary parts of another. In the illustrative embodiment, $W^0{}_9 = -W^4{}_8$; $W^1{}_9 = -W^5{}_8$; $W^2{}_9 = -W^6{}_8$; and $W^3{}_9 = -W^7{}_8$. Since each twiddle factor can be used for either of two values, the memory needed to store the twiddle factor values is decreased by a factor of 2.

Figure 6 illustrates an 8-point radix-2 FFT signal flow diagram 600 according to the presently disclosed FFT method. The FFT signal flow diagram 600 includes 3 stages, 602, 604, 606, and each stage has four FFT butterfly calculators per stage. Thus, each stage has the presently disclosed FFT method has the identical geometry. In this way, the group loop and its associated overhead, have been eliminated. Thus, the disclosed technique eliminates the triple loop nesting structure of the traditional FFT method. As discussed above, the elimination of the control and overhead necessary for exiting and entering the group loop will concomitantly decrease the complexity of the memory address calculations needed to execute the program and thereby increases program efficiency. In addition in the presently disclosed FFT method, the input data 608 is not re-ordered, and the output data values for each stage are properly ordered by the use of the unity stride memory operations discussed above. Thus, the re-ordering that is necessary in the traditional FFT method is eliminated, along with, the additional memory read/write operations. The decrease in the complexity of the calculations needed for the memory read/write operations improves the efficiency of the operation of the presently disclosed method over the traditional FFT methods as well.

As shown in Figure 6, the first stage 602 uses four twiddle factors 612 all of which equal W^0_8. The second stage 614 uses one pair of twiddle factors 614, W^0_8 and W^2_8, twice in a row in that order. The third stage 606 uses four twiddle factors W^0_8, W^2_8, W^1_8, and W^3_8 in that order.

Thus, the number of different twiddle factor values used per stage increases as a power of two and the twiddle factor values are retrieved from memory in a bit-reversed order. Therefore, in a preferred embodiment, the twiddle factors are stored in bit-reversed order to simplify the memory operations and increase efficiency. Figure 8 is a schematic illustration of the bit reversal process used to store four twiddle factors for the presently disclosed FFT technique.

In one embodiment, the FFT method is programmed for execution in a general-purpose computer using C, Java, or C++ or other suitable high level language. In another embodiment, the FFT method is programmed for use in a digital signal processing (DSP) system. In the DSP embodiment, the FFT method would use four pointer registers, and preferably the pointer register used to store the twiddle factors in bit reversed order is a circular address register. In addition, an out of place buffer placement of intermediate values is employed to eliminate instability within the inner, butterfly, loop.

As an example, for a 256-point radix-2 FFT, there will be 128 butterfly calculations in 8 stages. Assuming 3 cycles per butterfly, this will require a minimum of 128*8*2=3072 cycles. This number is not achievable however, because of the loop overhead and the overhead in the butterfly setup. Empirical tests have measured a traditional FFT method as using approximately 6600 cycles. The present FFT technique uses approximately 3330 cycles, i.e., nearly a 50% reduction in the number of cycles. Listed below is exemplary simulation code in the C programming language for one embodiment of the presently disclosed FFT method.

```
For (int s = 0; s < lgn; s++){                    Stage Loop
     For (int k = 0; s < no2; k++){               Butterfly Loop
       R0 = *i0++; r1 = i1++; r3 = i3++;
       real(r4) = real(r1)*real(r3)-imag(r1)*imag(r3);
       imag(r4) = real(r1)*image(r3)+image(r1)*real(r3);
       real(r5) = real(r0) + real(r4);
       imag(r5) = imag(r0) + imag (r4)
       real(r6) = real(r0) - real(r4);
       imag(r6) = imag(r0) - imag(r4);
       *i2++ = r5;
       *i2++ = r6;
     }
     p0 <<= 1; 13.L = p0;                     Unity Power Update
     i1.B = i2.B; i2.B = i0.B; i0.B = i1.B    Exchange
}.
```

Where lgn is the $\log_2(N)$. Accordingly, there is no group loop and the twiddle factors are updated in the unity power update step.

Those of ordinary skill in the art will appreciate that variations to and modifications of the above-described FFT methods and apparatus may be made without departing from the inventive concept disclosed herein. Accordingly, the invention should be viewed as limited solely by the scope and spirit of the appended claims.